Constituency Parsing with the help of PCFGs

Danish Shaikh
M.Tech (ECE)
shaikhm@iisc.ac.in

Abstract

Parsing is a process of determining the syntactic structure of a text by analyzing its constituent words based on an underlying grammar of the language. Parsing can be used in Sentiment Analysis, Relation Extraction, Grammar checking, etc. The dataset that has been used is 10% of Penn Treebank from NLTK. This model extracts syntactic structure of the text which determines the degree to which the text gives meaningful information. The parser that is generated by this model is compared with some exiting parsers avalibale online viz.,
1. Berkeley's Parser. Link is here.
2. Standford's Parser. Link is here.
Github Link to the model is available here.

1 Introduction

Parsing involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. For example, lets consider a sentence ”John is playing game”. After parsing, it will be stated in terms of its constituents, as ”John”, ”is”, ”playing”, ”game”. Parsing in natural language is termed as to analyze the input sentence in terms of grammatical constituents, identifying the parts of speech, syntactic relations. Here, ”John”, ”is”, ”playing”, ”game”, are tokens for above sentence. English language provides us with eight part of speech, viz: article, noun, pronoun, verb, adverb, adjective, preposition, conjunction. For example, in the above sentence, ”John is playing game” part-of-speech for each token is ”noun” for ”John” and ”game”, ”verb” for ”is” and ”playing”. But ambiguity is the main culprit to make this task harder. Example ”book”, it can be ”noun” or ”verb”, depending upon its use, parsing is use to find the correct (probabilistically) parse for a word or a sentence. The parsing process is shown in Figure 1.

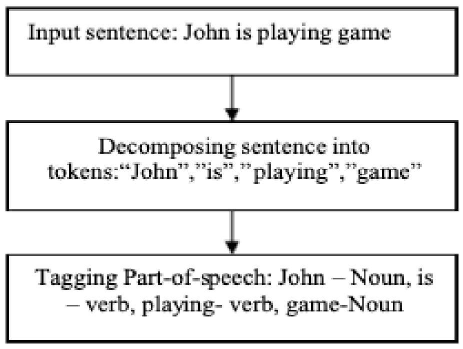
Input sentence: John is playing game
Decomposing sentence into tokens:“John”,”is”,”playing”,”game”
Tagging Part-of-speech: John – Noun, is – verb, playing- verb, game-Noun

Figure1. Parsing process

2 Parsing Techniques

Parsing results in generation of parse tree. Parse tree gives a connection between the sentence and the grammar, which describes how the grammar was used to produce the sentence. Natural Language processing has two basic parsing techniques: Top-Down Approach and Bottom-Up Approach. The implemented algorithm uses Cocke-Kasami-Younger parser (Bottom-Up approach) to generate the parse tree of new sentences.

2.1 Cocke-Kasami-Younger Parser

It is a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy. Intermediate solutions are stored and only intermediate solutions that contribute to a full parse are further pursued.

2.2 Constituency Parser

Constituency parsing aims to extract a constituency-based parse tree from a sentence that represents its syntactic structure according to a phrase structure grammar.
For Instance:

```
              Sentence (S)
                   |
   +---------------+-------------+
   |                             |
Noun (N)                 Verb Phrase (VP)
   |                             |
John                    +--------+--------+
                        |                 |
                    Verb (V)           Noun (N)
                        |                 |
                      sees              Bill
```

2.3 Probabilistic Context-Free Grammar

A Probabilistic Context-Free Grammar (PCFG) is simply a Context-Free Grammar with probabilities assigned to the rules such that the sum of all probabilities for all rules expanding the same non-terminal is equal to one. PCFG parsing takes advantage of probabilities by giving you the most probable parse for a sentence. However, PCFGs are still not powerful enough to describe context-sensitive languages. Example:

$S \rightarrow NP\ VP$	1	$Det \rightarrow$ the	1.0
$NP \rightarrow Det\ N$	0.7	$N \rightarrow$ girl	0.4
$NP \rightarrow NP\ PP$	0.3	$N \rightarrow$ boy	0.4
$VP \rightarrow V\ NP$	0.6	$N \rightarrow$ telescope	0.2
$VP \rightarrow V\ NP\ PP$	0.4	$V \rightarrow$ saw	1.0
$PP \rightarrow P\ NP$	1	$P \rightarrow$ with	1.0

3 Task:

Problem Definition: To implement parser for any given sentence by generating grammar rules with probabilities from the dataset.
Contribution: Due to the limited dataset, it is highly unlikely to cover all the words in the English dictionary. To overcome this issue, I have used Add-1 smooting and added 'UNK' tags for all the Non-Termimal to Terminal rules.

4 Implementation:

NLTK provides access to 10% sample of the Penn Treebank, which is our dataset for this model. It consists of 3914 sentences. The dataset is splitted into 80%-20% for creating training and validation datasets.

For training:

- Step 1: Imported dataset from nltk.
- Step 2: Using productions(), I have generated grammar rules from the dataset.
- Step 3: Parameters can be directly estimated from frequency counts in the treebank.

$$P(\alpha \rightarrow \beta \mid \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{count}(\alpha \rightarrow \gamma)} = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}$$

- Step 4: But as smoothing is the main criteria here, due to the limited dataset, I have used Add-1 smoothing so as to add "UNK" tag probabilities.
- Step 5: Extracted Non-Terminal to Terminal Rules and added "UNK" tags
- Step 6: Calculated probabilities with the help of Add-1 smoothing.
- Step 7: Dumped all the rules with probabilities into pickle file and thereby saving the model.

For Testing:

- Step 1: Read the model that has been generated from the training phase.
- Step 2: Tokenize the test sentence and store into a list.
- Step 3: Replace words that are not in our grammar with "UNK" tags. This is done with the help of check_coverage().
- Step 4: Generate trees with our gramamr using CKY-algorithm.
- Step 5: Print all possible trees.

4.1 Assumptions

1. We have assumed all the words in a sentence appear independently.
2. As our grammar model doesn't take into account the context, it is same for all the test sentences.
3. We have used this limited dataset along with "UNK" tags will cover all the sentences in English language.

5 Evaluation of the Model:

We have used 3 metrics to evaluate our model viz., Labeled Precision, Labeled Recall, F1. Results are tabulated below:

Metrics	Scores
Labelled Precision	68.5
Labelled Recall	65.5
F1 Score	66.96

Table 1: Metrics calculated on the validation dataset

6 Problem of Add-1 Smoothing in PCFG

Add-1 smoothing is very crude way to avoid zeros.
As our model is highly dependent on this grammar rule, we can't guarantee the correctness of the generated parser.

7 Analysis

The generated parser of the model is compared with two parsers available online.
1. Berkeley's Parser. Link is here.
2. Standford's Parser. Link is here.

For short sentence 1: 'The chicken is ready to eat.'
Ambiguity: Someone cooked chicken and it is ready to eat, or the chicken itself is ready to eat earthworms (or something).

1. Our Model:
   (S (NP-SBJ-105 (DT The) (NNS chicken)) (VP (VBZ is) (PP-PRD (JJ ready) (PP-PRD—⟨ TO-NP ⟩ (TO to) (NP (RBS eat.)))))) (p=1.79091e-22)

2. Berkeley's:
   (S (NP (DT The) (NN chicken)) (VP (VBZ is) (ADJP (JJ ready) (S (VP (TO to) (VP (VB eat)))))) (. .)))

3. Stanford:

   (S (NP (DT The) (NN chicken)) (VP (VBZ is) (ADJP (JJ ready) (S (VP (TO to) (VP (VB eat))))))

For short sentence 2: 'Landmine claims dog arms company.'
Ambiguity: A landmine claimed that a dog was providing weapons to a company, or a landmine laid claim to a company producing weapons.

1. Our Model:
   (S (NP-SBJ (DT Landmine) (NP-SBJ—⟨NNS-CC⟩ (NNS claims) (NP-SBJ—⟨CC-NNS⟩(CC dog) (NNS arms)))) (ADJP-PRD (UH company.))) (p=2.75089e-18)

2. Berkeley's:

   (S (NP (NNP Landmine)) (VP (VBZ claims) (NP (NN dog) (NNS arms) (NN company))) (. .)))

3. Stanford:

   (S (NP (NNP Landmine)) (VP (VBZ claims) (NP (NN dog) (NNS arms) (NN company))) (. .)))

For long sentence 1: 'The cat chased the mouse until it stumbled and fell.'
Ambiguity: Which one fell, the cat or the mouse?

1. Our Model:
   (S (NP-SBJ (DT The) (PRP cat)) (S—⟨ADJP-PRD-ADVP⟩ (ADJP-PRD (JJR chased) (PP (ADVP (DT the) (RBR mouse)) (PP—⟨IN-NP-LGS⟩(IN until) (NP-LGS (PRP it))))) (ADVP (RBR stumbled) (ADVP—¡CC-RBR¿ (CC and) (RBR fell.))))) (p=5.71833e-30)

2. Berkeley's:
   (S (NP (DT The) (NN cat)) (VP (VBD chased) (NP (DT the) (NN mouse)) (SBAR (IN until) (S (NP (PRP it)) (VP (VP (VBD stumbled)) (CC and) (VP (VBD fell)))))) (. .)))

3. Stanford:
   (S (NP (DT The) (NN cat)) (VP (VBD chased) (NP (DT the) (NN mouse)) (SBAR (IN until) (S (NP (PRP it)) (VP (VP (VBD stumbled)) (CC and) (VP (VBD fell)))))) (. .)))

For long sentence 2: 'My mother never made chocolate cake, which we all hated.'
Ambiguity: Did we hate chocolate cake, or did we hate that our mother never made it for us?

1. Our Model:
   (S (NP-SBJ (PRP$ My) (NN mother)) (S—⟨ADJP-PRD-PP⟩ (ADJP-PRD (RBR never) (VBN made)) (PP (JJ chocolate) (PP—⟨TO-NP⟩ (TO cake,) (NP (NP (WDT which)) (NP—⟨NP-ADV-ADVP⟩ (NP-ADV (PRP we)) (ADVP (DT all) (RBR hated.)))))))) (p=2.01951e-36)

2. Berkeley's:
   (S (NP (PRP$ My) (NN mother)) (ADVP (RB never)) (VP (VBD made) (NP (NP (NN chocolate) (NN cake)) (, ,) (SBAR (WHNP (WDT which)) (S (NP (PRP we) (DT all)) (VP (VBN hated)))))) (. .)))

3. Stanford:
   (S (NP (PRP$ My) (NN mother)) (ADVP (RB never)) (VP (VBD made) (NP (NP (NN chocolate) (NN cake)) (, ,) (SBAR (WHNP (WDT which)) (S (NP (PRP we)) (ADJP (RB all) (VBN hated)))))) (. .)))

8 Conclusion

The implemented model gives parsing of any sentence, including the words that are not present in the grammar, with **labeled precision = 68.5** , **labeled recall = 65.5** and **F1 score = 66.96.**

9 References

1. NLTK Documentation. Link is here.
2. Basic parsing techniques in natural language processing. Link is here.
3. Treebanks and PCFGs. Link is here